FUNKCJE

1. Funkcja (read), (readline), (test ...)

Funkcja (read) służy do czytania danych z klawiatury. Czyta ona tyko JEDNO POLE (tzn. do pierwszej spacji). Aby odczytać kilka słów, musimy otoczyć je znakiem " " (wtedy funkcja (read) potraktuje je jako łańcuch, czyli jedno pole). Po (read) nie jest potrzebny dodatkowy "crlf", ponieważ funkcja (read) po odczytaniu danej umieszcza kursor w nowej linii.

Funkcja (readline) służy do czytania z klawiatury większej liczby pól naraz.
Oto przykład jej działania:

```
(defrule test-komendy-readline
(initial-fact)
=>
(printout t "Napisz dowolny tekst umieszczony w nawiasie (wiecej niz 1 wyraz) :")
(bind ?string (readline))
(assert (string ?string))
(assert-string ?string))
```

CLIPS> (reset)
CLIPS> (run)
Napisz dowolny tekst umieszczony w nawiasie (wiecej niz 1 wyraz) : (Ala ma kota)
CLIPS> (facts)
f-0 (initial-fact)
f-1 (string "(Ala ma kota)" ;ten fakt został potwierdzony jako jednopolowy łańcuch
f-2 (Ala ma kota) ;ten fakt jest faktem wielopolowym

Funkcja (assert-string) potwierdza łańcuch jako fakt, czyli łańcuch zostaje rozdzielony na pola, które staja się polami faktu wielopolowego. Nawias wokół tekstu jest konieczny jedynie ze względu na składnię funkcji (assert-string ...). Bez tego nawiasu wystąpi błąd w czasie wykonywania programu.

Funkcja (test ...) porównuje liczby, łańcuchy i zmienne. Umieszczamy ją ZAWSZE PO LEWEJ STRONIE REGUŁY. Jej składnia jest następująca:

(test (funkcja-porównująca argumenty))

gdzie:
funkcja porównująca jest jedną z poniższych funkcji, a argumenty to 1 lub więcej argumentów żądanych przez funkcję:

=	równe
<>	różne
>=	większy równy
>	większy
<=	mniejszy lub równy
<	mniejszy

Wewnątrz funkcji (test ...) można używać operacji arytmetycznych, oczywiście zapisanych w notacji CLIPS'a. Reguła, która ma po lewej stronie funkcję (test ...), zostanie uaktywniona tylko wtedy, gdy będzie spełniony warunek zawarty w tej funkcji, np. reguła zawierająca warunek: (test (> ?delta 0)) będzie uaktywniona, gdy zmienna ?delta ma wartość >0, a ponadto będą istniały wszystkie fakty konieczne dla uaktywnienia tej reguły.

Przykład użycia funkcji test:

```
(defrule start
     (initial-fact)
=>
     (printout t "Podaj a :")
     (bind ?a (read))
     (printout t "Podaj b :")
     (bind ?b (read))
     (printout t "Podaj c :")
     (bind ?c (read))
     (assert (a ?a)(b ?b)(c ?c)))

(defrule sumuj
     (a ?a)(b ?b)(c ?c)
=>
     (bind ?suma (+ ?a ?b ?c))
     (printout t "Suma a+b+c = " ?suma crlf)
     (assert (suma ?suma)))

(defrule test1
     (suma ?suma)
     (test (= ?suma 0))
=>
     (printout t "Suma jest rowna zero" crlf))

(defrule test2
     (suma ?suma)
     (test (< ?suma 0))
=>
     (printout t "Suma jest mniejsza od zera" crlf))

(defrule test3
     (suma ?suma)
     (test (> ?suma 0))
=>
     (printout t "Suma jest wieksza od zera" crlf))
```

2. Funkcje logiczne AND, OR i NOT po lewej stronie reguły.
Wszystkie dotychczas przez nas pisane reguły miały po lewej stronie niejawną funkcję logiczną AND, tzn. dla uaktywnienia reguły musiały być spełnione wszystkie warunki.

Zapis:
(defrule nazwa-reguly
(warunek1)(warunek2)...(warunekN)
=>
....
)

jest więc równoważny zapisowi:
(defrule nazwa-reguly
(and (warunek1)(warunek2)...(warunekN))
=>
....
)

Po lewej stronie reguły możemy używać funkcji logicznej OR, gdy chcemy, aby reguła została uaktywniona w razie spełnienia któregokolwiek z warunków po lewej stronie:

```
(defrule nazwa-reguly
        (or (warunek1)(warunek2)...(warunekN))
=>
....
)
```

lub funkcji logicznej NOT, gdy chcemy aby reguła została uaktywniona, gdy warunek nie jest spełniony, (tzn. warunkiem jest tu nie wystąpienie faktu-1):

```
(defrule nazwa-reguly
        (not (fakt-1))
=>
....
)
```

Uwagi:
- gdy chcemy zanegować kilka warunków, do każdego musimy użyć osobnej funkcji (not ...)
- konstrukcja typu (not (not (fakt-1))) jest niedozwolona.

3. Funkcje definiowane przez użytkownika

Użytkownik CLIPS'a może definiować własne funkcje za pomocą komendy (deffunction ...). Tak zdefiniowana funkcja jest widziana globalnie przez wszystkie reguły, co pozwala uniknąć pisania w wielu regułach tych samych akcji. Składnia (deffunction ...):

```
(deffunction nazwa-funkcji
        (?arg1  ?arg2   ...        ?argM) ;lista argumentów
        (akcja1
        akcja2
        ....
        akcjaK))                        ; TYLKO OSTATNIA AKCJA ZWRACA WARTOŚĆ !
```

Argumenty są argumentami pozornymi (parametrami) (są widziane tylko wewnątrz funkcji), co oznacza, że nazwy argumentów nie kolidują z nazwami zmiennych w regule, nawet jeżeli są to takie same nazwy. Tylko ostatnia akcja będzie zwracać wartość. Komenda (printout t ...) zostanie wykonana nawet gdy nie będzie ostatnia akcja, ponieważ wydruk jest tylko ubocznym efektem działania komendy (printout t ...). Poniższy przykład pokazuje, że funkcja zwraca tylko ostatnią akcję:

```
(deffunction ostatnia-akcja
(?a ?b)         ;argumenty
(printout t "Wartosci wejsciowe: " ?a " " ?b crlf)
(+ ?a ?b) ;ta akcja nie zwraca wartości
(- ?a ?b) ;ta również
(* ?a ?b))      ;ostatnia akcja zwraca wartosc
```

Funkcja użytkownika może zostać użyta jako argument innej funkcji np. (sqrt ...), jak w poniższym przykładzie:

Przykład

```
(defrule start
      (initial-fact)
=>
      (printout t "Podaj dlugosc boku a : ")
      (bind ?bok_a (read))
      (printout t "Podaj dlugosc boku b : ")
      (bind ?bok_b (read))
      (assert (bok_a ?bok_a)(bok_b ?bok_b)))

(deffunction pole-prostokata
(?a ?b)
(* ?a ?b))

(defrule oblicz
      (bok_a ?bok_a)(bok_b ?bok_b)
=>
(printout t "Pole = " (pole-prostokata ?bok_a ?bok_b) crlf)
(printout t "Pierwiastek z pola = " (sqrt(pole-prostokata ?bok_a ?bok_b)) crlf))
```

Zadania:

1.Napisać program obliczający przy użyciu funkcji wartość *delty* równania kwadratowego postaci:

$$a \cdot x^2 + b \cdot x + c = 0$$

2.Napisać program obliczający przy użyciu funkcji *pole* koła o promieniu *r*:

$$pole = \pi \cdot r^2$$

3.Napisać program obliczający przy użyciu funkcji sumę *n* wyrazów ciągu geometrycznego, korzystając z zależności:

$$S_n = \frac{a_1 \cdot \left(1 - q^n\right)}{1 - q}$$

gdzie:

S_n – suma wyrazów ciągu
a_1 – pierwszy wyraz ciągu
q – iloraz ciągu ($q = \frac{a_2}{a_1}$)

Użytkownik ma podać dwa pierwsze elementy ciągu geometrycznego a_1 i a_2, oraz liczbę *n* mówiącą o tym z ilu elementów ciągu ma być liczona suma.

4.Napisać program obliczający przy użyciu funkcji *n*-ty wraz ciągu Fibonacciego, korzystając z zależności:

$$F_n = \frac{1}{\sqrt{5}} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} \cdot \left(\frac{1-\sqrt{5}}{2}\right)^n$$